Remarks

Status of application

Claims 1-60 were examined and stand rejected in view of prior art, as well as stand rejected for technical issues. In view of the amendments to the specification and claims and the remarks made herein, reexamination and reconsideration are respectfully requested.

The invention

Applicant's invention assists users in refactoring multiple software modules or programs residing in different projects. The solution automates the process of refactoring of a first module (e.g., a library) at one point in time and applying the same refactoring to other module(s) (e.g., an application dependent on the library) at a later time. The refactoring process is automated so as to avoid the need to manually refactoring the multiple interrelated modules, which can be tedious and error prone.

Applicant's approach for asynchronous refactoring of two (or more) dependent modules can be summarized as follows. When an original refactoring of a first module residing in a first project is performed, meta information regarding the refactoring is recorded (e.g., tracked and persistently stored). The meta information that is recorded about the refactoring includes particular symbol(s) affected by the refactoring and the specifics of the changes that are made. Subsequently, when a second software module residing in a second project is refactored, Applicant's solution performs the refactoring based on the recorded meta information. The recorded meta information is applied to the second module(s) in two phases or sets of operations. In the first phase, entries for one or more symbols are copied (injected) into the symbol table (e.g., compiler symbol table) used for building the second module (e.g., application) based on the recorded refactoring meta information. In the second phase, the actual refactoring of the dependent module(s) is performed, using a conventional refactoring on the dependent module(s) that were fixed up in the first phase.

General

A. Declaration

The Examiner has objected to the declaration filed by Applicant as being defective on the basis that the last name of inventor Mark K. Howe is misspelled as "Howe". Applicant respectfully points out the copy of the declaration filed with the Application was transmitted by telefax and that the "misspelling" is obviously an artifact resulting from fax transmission. In order to address the Examiner's objection, Applicant is filing simultaneously with this amendment a scanned in copy of the original declaration (and power of attorney), thereby overcoming the objection.

B. Amendments to Specification

In accordance with the request of the Examiner, Applicant has amended the specification to include several references for the trademark Java. However, Applicant takes no position as to the validity or ownership of any such trademarks.

C. Section 101 rejection

Claims 16-30 stand rejected under 35 U.S.C. 101 on the basis of non-statutory subject matter on the basis that Applicant's claimed system constitutes computer software alone without the necessary physical components (hardware) to constitute a machine or manufacture and hence are non-statutory. Although Applicant respectfully believes that the Examiner has incorrectly construed Applicant's specification and claims as stating that the elements of Applicant's invention can only be implemented in software, Applicant has amended claim 16 by adding claim limitations of a computer system having a processor and memory. These claim limitations find support in Applicant's specification which expressly state that elements of Applicant's invention may be implemented in hardware, software or firmware (or combinations thereof). This is expressly stated, for example, at paragraph [44] of Applicant's specification as follows: "...the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof" (Applicant's specification, paragraph [44], emphasis added). Applicant's specification also describes in detail a computer hardware and software environment in which Applicant's invention may be implemented (Applicant's specification, paragraphs [45]-[55]). As Applicant's claim defines a useful machine or item of manufacture in terms of a hardware or hardware and software combination,

Applicant respectfully believes that it defines a statutory product and overcomes the rejection of claims 16-30 under Section 101.

Prior art rejections

A. Section 102(b): Li

Claims 1-60 stand rejected under 35 U.S.C. 102(b) as being anticipated by Li, et al "Tool Support for Refactoring Functional Programs", Proceedings of the 2003 ACM SIGPLAN workshop on Haskell, Uppsala, Sweden, August 28, 2003 (hereinafter "Li"). The Examiner's rejection of Applicant's claim 1 as follows is representative of the Examiner's rejection of these claims as anticipated by Li:

Per Claim 1:

The Li publication discloses:

- in response to a change that affects a particular symbol of a software module that resides in a first project, refactoring the software module of the first project to propagate the change to all instances of the particular symbol in the software module ("... Figure 6 also shows a particular refactoring scenario. The user has selected the identifier format in the definition of table, has chosen the duplicateDef command from the Refactor menu ... the refactorer ensures consistent renaming, including recursive calls ..." on pg. 34, 1st column, par. 2, lines 1-9)

- during the refactoring of the software module of the first project, recording meta data about the refactoring that is required to effect the change; and automatically propagating the change to a dependent software module residing in a second project, by refactoring the dependent software module based on the recorded meta data about the refactoring that occurred to the software module of the first project ("... Renaming ... Any program identifier can be renamed, as in Figure 4 where fmt replaces format. It is important to ensure that all and only the uses of one binding are renamed. Conditions. The existing binding structure must not be affected. No binding for the new name may exist in the same binding group. No binding for the new name may intervene between the binding of the old name and any of its uses, as the renamed identifier would be captured by the renaming. The binding to be renamed must not intervene between existing bindings and uses of the new name. ..." on pg. 30, 2nd column, par. 9 to par. 10; and see Figure 6, item "rename" under the menu "Refactor").

Under Section 102, a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in the single prior art

reference. (See, e.g., MPEP Section 2131.) As will be shown below, Li fails to teach each and every element set forth in Applicant's claims 1-60, and therefore fails to establish anticipation of the claimed invention under Section 102.

At the outset, Applicant does not claim to have invented the notion of refactoring a software program. Applicant acknowledges that Li (as well as other prior art references) provide a solution for refactoring a software program in a single project. However, Applicant's claimed invention provides an improved refactoring solution which enables two different programs (or modules) that reside in different projects to be refactored asynchronously (Applicant's specification, paragraph [70]). For instance, a given program such as an application program may be dependent upon another program such as a library, with the application and the library each implemented in a different project and even in a different programming language (Applicant's specification, paragraph [25]; see also paragraphs [71]-[78]). Applicant's solution automates the process of refactoring the library (first module) at one point in time and applying the same changes to the application (second module) at a later point in time so as to maintain the compatibility between the two (Applicant's specification, paragraph [76]). Significantly, Applicant's invention enables the refactoring of the two modules to be performed asynchronously (i.e., at two different points in time). For example, after a library has been refactored, an application which is dependent on the library may not compile and run properly against the refactored library (Applicant's specification, paragraph [76]). However, Applicant's invention enables the same refactoring to be applied to the application in an automated and consistent fashion (Applicant's specification, paragraphs [77]-[78]).

It should be noted that the actual refactoring of the library (first module) itself could be performed using a refactoring solution such as that described by Li. However, Applicant's invention takes additional steps by recording meta information during the refactoring of the first module (e.g., library) in a first project, so that the same refactoring can be applied against module(s) in other projects (e.g., the application which uses the library but which resides in a different project) at a later point in time (Applicant's specification, paragraph [70]). The meta information that is recorded about the refactoring includes particular symbols being affected by the refactoring and the specifics

of the changes that are made (Applicant's specification, paragraph [71]). Subsequently, when the second module (e.g., application) is refactored, Applicant's solution uses the recorded meta information for refactoring the second module (Applicant's specification, paragraph [71]). Applicant's claims have been amended in an effort to more clearly articulate these distinctive features of refactoring modules residing in two different projects at two different times (i.e., asynchronously). For example, amended claim 1 includes claim limitations of:

> A method for refactoring a plurality of interdependent software modules that reside in separate projects, the method comprising:
> in response to a change that affects a particular symbol of a software module that resides in a first project, refactoring the software module of the first project to propagate the change to all instances of the particular symbol in the software module;
> during the refactoring of the software module of the first project at a given point in time, recording meta data about the refactoring that is required to effect the change; and
> at a subsequent point in time, automatically propagating the change to a dependent software module residing in a second project, by refactoring the dependent software module based on the recorded meta data about the refactoring that occurred to the software module of the first project.
>
> (Applicant's amended claim 1, emphasis added)

Turning to Li, one finds that it does not describe a multi-project, asynchronous refactoring, but rather simply describes the performance of a conventional refactoring of a single program. The Examiner references Li at page 30, 2nd column, paragraphs 9-10 as providing teachings of recording meta data during refactoring of a module residing in a first project and using this meta data to refactor a dependent module residing in another project. However, Applicant's review of the referenced paragraphs of Li (as well as the balance of the reference) indicates that Li describes a rename refactoring of a program and makes no mention of applying the same refactoring to two different modules residing in different projects. In addition, Applicant's review of the Li reference finds no teaching of recording meta data during refactoring of the first module and using the recorded information at a later point in time to refactor the second module as provided in Applicant's specification and claims.

Further distinctions between Applicant's invention and Li's refactoring solution

can be found in Applicant's dependent claims. For example, Applicant's claim 2 includes claim limitations of copying (or injecting) symbol information about the refactoring of the first module (of a first project) into a symbol table of the second module (in a second project) for performing the refactoring of the second module. Here, Applicant's approach provides for applying meta information recorded during refactoring of the first module to the second module in a two phase refactoring process (Applicant's specification, paragraph [72]). In the first phase, symbols are injected into the compiler symbol table used for building the second module(s) based on the recorded refactoring meta information (Applicant's specification, paragraphs [72]-[73]). In the second phase, the actual refactoring of the second module (e.g., application) is performed (Applicant's specification, paragraph [74]). At most, Li's conventional refactoring approach could be considered somewhat similar to the second phase of this process. However, Li includes no teaching of injecting symbols derived from a first refactoring into a compiler symbol table of the second module for performing this refactoring. For one thing, Li make no mention of a multi-project refactoring and, therefore, provides no teaching of copying symbol information into another project. Li does make mention of the fact that AST position information may be out of date as a result of a refactoring which may require rebuilding the AST (Li, page 35); however this is not analogous to Applicant's claim limitations of copying symbol information from a refactoring in one project into the symbol table used for refactoring another software module in another project.

All told, Li describes a conventional solution for refactoring modules of a given program which reside in a single project and are refactored at a single point in time. However, Li does not describe recording and storing meta information about this refactoring for subsequent use in performing a refactoring of another module(s) in a second project at a later point in time. Moreover, Li does not include the specific teachings of injecting symbol information derived from the recorded meta information into a symbol table used for refactoring the module in the second project as provided in Applicant's specification and claims. Therefore, as Li fails to teach each and every element set forth in Applicant's claims 1-60, Applicant respectfully believes that it does not anticipate Applicant's claimed invention under Section 102.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

Conclusion

In view of the foregoing remarks and the amendment to the claims, it is believed that all claims are now in condition for allowance. Hence, it is respectfully requested that the application be passed to issue at an early date.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 925 465-0361.

Respectfully submitted,

Date: November 29, 2007

/G. Mack Riddle/

G. Mack Riddle, Reg. No. 55,572
Attorney of Record

925 465-0361
925 465-8143 FAX